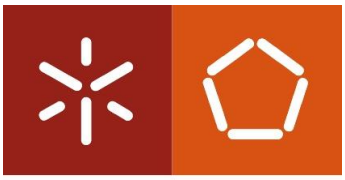

Universidade do Minho

Departamento de Informática

Sistema de Gestão de Vendas

Grupo Nº 56

João Teixeira (A85504)

Emanuel Rodrigues (A84776)

José Ferreira (A83683)

9 de Abril de 2019

Conteúdo

Capítulo 1

Introdução

O objetivo deste projeto é construir um sistema de gestão de vendas modular, de forma a ser capaz de armazenar informações de vendas e relacionar produtos, clientes e vendas de forma eficiente, aplicando uma arquitetura *Model, View, Controller* e conhecimentos sobre algoritmos e estruturas de dados. Como objetivo, é também necessário garantir o encapsulamento dos dados armazenados de forma a que não sejam possíveis alterações indevidas por agentes externos.
Ao longo deste relatório vamos descrever as nossas abordagens a estes problemas e apresentar alguns testes de performance do nosso projeto final.

Capítulo 2

Problema

Três ficheiros são fornecidos contendo informação sobre as transações de uma distribuidora:

- O primeiro contém ids de produtos.
- Outro, ids de clientes.
- O último integra informações sobre cada venda efetuada ao longo de um ano.

Com base nesses ficheiros é preciso responder a 12 querys fornecidas:

1. Ler os ficheiros acima referidos validando cada linha.
2. Listar todos os produtos cujo ID comece por uma dada letra.
3. Total de vendas e total faturado num dado mês por um dado produto, distinguindo os totais N e P, global ou filial a filial.
4. Listar códigos de produto que ninguém comprou.
5. Listar clientes que compraram em todas as filiais.
6. Número de clientes e de produtos não referidos nas vendas.
7. Tabular o número de produtos comprados por um dado cliente, mês a mês e por filial.
8. Total de vendas num dado intervalo temporal.
9. Listar clientes que compraram um dado produtos numa dada filial, distinguindo compras N de P.
10. Listar os produtos mais comprados por um dado cliente num dado mês, por ordem descendente.
11. Listar os N produtos mais vendidos, com número de clientes e número de unidades vendidas, em cada filial.
12. Listar os 3 produtos em que um dado cliente gastou mais dinheiro.

Capítulo 3

Módulos e API

3.1 Cliente

O módulo de Cliente tem uma API capaz de lidar com a informação de um cliente individual e respetiva informação relativa a onde efetuou compras, com vista à melhor resposta de queries que envolvam tal informação.

3.2 Clientes

O módulo de Clientes tem uma API capaz de lidar com a informação de todos os clientes, respetivo armazenamento e pesquisa.
Como estrutura principal de armazenamento dos Clientes individuais, após testes testar várias estruturas, optamos por escolher Hashtables da biblioteca GLib, da Gnome.

3.3 Produto

À semelhança do módulo de Cliente, o módulo de Produto tem uma API desenhada para tratar da informação referente a um produto individual e informação de onde foi vendido.

3.4 Produtos

No módulo de Produtos, a API está construida de maneira a que seja capaz de lidar, de forma eficiente com o armazenamento e pesquisa de todos os produtos.
Para este módulo, como no módulo de Clientes, decidimos utilizar as mesmas Hashtables da biblioteca GLib da Gnome.

3.5 Venda

No módulo de Venda, temos uma API capaz de dar parse de uma string com o formato previamente definido, colocar numa estrutura com os campos necessários de maneira a preservar a informação, para posteriormente ser tratada pelo módulo de Filiais e Faturação.

3.6 Fatura

Neste módulo, a API está definida de forma a conseguir, a partir de uma Venda, criar/atualizar uma fatura, contendo esta, a faturação relativa a um produto.

3.7 Faturas

O módulo de Faturas é capaz de comportar informação sobre toda a faturação, organizada por produtos e guardada numa Hashtable. Uma estrutura do tipo Faturas para além de guardar a faturação individual de cada produto, para um mais rápido acesso, guarda também valores totais de faturação e número de vendas.

3.8 Filial

Para o módulo de filial, está definida uma API capaz de fazer a ligação entre clientes e os respetivos produtos por eles comprados, e vice-versa.
Uma estrutura do tipo Filial é composta por duas Hashtables, uma que contém informação relativa a todos os clientes que fizeram compras na dada filial, que produtos compraram e respetiva faturação, e uma segunda que contém informação relativa a todos os produtos vendidos na dada filial, mais concretamente que clientes adquiriram o produto.

3.9 SGV

O módulo SGV é o módulo que junta todos os acima descritos, contendo este uma estrutura que contém um Catálogo de Produtos, um Catálogo de Clientes, uma estrutura Faturação e um array de três Filiais. Este módulo faz a ponte entre todos os módulos internos e o exterior, sendo este o módulo que ao qual o controlador faz pedidos, e tem métodos capazes de responder a todas as queries pedidas.

3.10 Controller

Com o objetivo último de facilitar a pesquisa de queries, estas foram divididas em 3 categorias, Clientes, Produtos e Vendas. Para tornar a pesquisa ainda mais *streamlined*, algumas queries foram unidades, visto que a informação obtida está proximamente relacionada. Assim, no módulo do Controller estão os menus de escolha de categoria e os menus de cada uma das categorias.

3.11 QueryUI

O módulo QueryUI contém um menu para cada query fornecida, assim como os menus que unem várias queries.

3.12 View

O módulo View contém métodos de impressão genérica de estruturas. Entre elas, apresentação de listas de strings sobre a forma de páginas navegáveis que se ajustam automaticamente ao número de strings fornecidas.

3.13 Formating

O módulo Formating contém um conjunto de strings que quando impressas no terminal do *Linux* alteram as propriedades do texto escrito, tais como a cor e a sua formatação. Também contém o tamanho máximo do nome de um ficheiro em *Fat32* a fim de ser utilizado em buffers de leitura de nomes de ficheiro.

3.14 Argv

Este módulo permite realizar *benchmarks* a queries através dos argumentos da linha de comandos. Para tal várias funcionalidades foram implementadas:

1. `./SGV 1`: Cronometrar todas as queries com parâmetros default.
2. `./SGV 2`: Cronometrar a query 2, por exemplo, com parâmetros default.
3. `./SGV 2 A`: Cronometrar a query 2, por exemplo, com parâmetro A.
4. `./SGV 8 3 5`: Cronometrar a query 8, por exemplo,no intervalo de 3 a 5.

```
$ ./SGV 8 3 5
CPU Time used to Load Files:3.447456
CPU Time used to Answer Query:0.000012
CPU Time used to Free structs:1.173244
```

Capítulo 4

Arquitetura do projeto

Este projeto segue uma Arquitetura do tipo *MVC* (Modelo, Apresentação e Controlador).

4.1 Modelo

O modelo tem como base dois grandes módulos, o módulo Filial e o módulo Faturação, e têm como módulos auxiliares ambos os Catálogos de Produtos e de Clientes.
Estes dois grandes módulos estão unidos num módulo mais geral, SGV, e este recebe os pedidos do Controlador, e trata a informação contida em todos os módulos, de forma a devolver-lhe a informação pedida da forma mais eficiente. É nesta camada onde está toda a parte de algoritmos e dados, nunca esta conhecendo ou dada a conhecer à camada da Apresentação.

4.2 Controlador

O Controlador tem como base dois grandes módulos, o módulo queryUI e o módulo controller. O controller contém os menus de escolha das categorias e os menus de escolha de queries. O queryUI contém os menus para cada query individual.

4.3 Apresentação

A apresentação contém apenas um módulo, o view. Este contém funções de apresentação genérica, tais como funções para apresentar array de strings e funções para apresentar matrizes de inteiros.

Capítulo 5

Estruturas de dados

Como descrito no capítulo 3, todas as nossas estruturas de dados consistem em Hashtables da GLib, escolha feita após múltiplos testes empíricos de abordagens distintas.
Numa abordagem inicial, na primeira fase do projeto, utilizamos arrays para guardar os dados. Para esta abordagem, observamos tempos de leitura e validação de ficheiros superiores a 5 minutos.
Numa segunda abordagem, no inicio da segunda fase, optamos por utilizar Árvores Auto-Balanceadas da biblioteca GLib, onde os tempos de leitura e validação dos ficheiros baixaram para cerca de 7 segundos, graças à pesquisa $O(logN)$, operação mais utilizada na validação das vendas.
Como abordagem final, implementamos Hashtables, também da biblioteca GLib. Nesta, os valores não são armazenados de forma ordenada. No entanto, tal como concluímos com esta versão, com ferramentas tão otimizadas para a ordenação como o *qsort()* da linguagem de programação C, a inserção ordenada acaba por ter um impacto demasiado negativo nos tempos de inserção.
Por um lado, graças ao tempo de inserção e pesquisa $O(1)$, conseguimos reduzir o tempo de carregamento de ficheiros para menos de 4 segundos, como descrito na tabela C.1, um melhoramento de aproximadamente 40%.
Por outro lado, graças à implementação otimizada das Hashtables na biblioteca em uso, percorrer todos os elementos desta, de acordo com os nossos testes, é tão rápido como na implementação anterior de AVLs. Tal resulta num tempo de resposta a queries inalterado quando comprado com a implementação anterior.
Assim, com a utilização de Hashtables conseguimos uma melhoria de aproximadamente 40% nos tempos de *load* e manter os mesmos resultados nos tempos de resposta a queries.

Capítulo 6

Testes e Benchmarks

Durante a execução do nosso projeto, foram efetuados diversos testes ao código, com ferramentas como *Valgrind*, para verificar a presença de *memory leaks*, *memusage*, para fazer proffiling à memoria utilizada, *gprof* para ver quais funções pesam mais na execução do programa, com vista a otimizar as partes mais lentas, e para medição dos tempos de execução utilizamos a biblioteca *time.h*.

6.1 Memória

Utilizando a ferramenta *Valgrind*, conseguimos verificar que o programa corre sem *memory leaks*, à exceção de algumas causadas pela biblioteca *glib*.

Com a ferramenta *memusage*, obtivemos o gráfico de uso de memória presente na figura B.1, concluindo que as nossas estruturas preenchidas com os dados dados, estão a consumir, em pico, por volta dos 360MB.

6.2 Tempos de execução

Com recurso à biblioteca de C *time.h*, efetuamos alguns benchmarks, obtendo a tabela C.1, com os tempos médios de execução, com vários números de vendas, concluindo assim que os tempos das queries são independentes do número de vendas, tendo este fator apenas impacto nos tempos de load.
Correndo a ferramenta *gprof*, observamos que a função onde é gasto mais tempo é a função que cria faturas, pois esta tem cálculos demorados, o que levou a que não a conseguíssemos otimizar.

Capítulo 7

Conclusão

Para concluir, conseguimos cumprir todos os requisitos propostos, conseguindo implementar todos os módulos e estrutura-los como pedido, sendo assim capaz de responder a todas as queries da forma que nos pareceu mais eficiente.
Como trabalho futuro, gostaríamos de conseguir melhorar os tempos de load de ficheiros, no caso do ficheiro de um milhão de vendas, para tempos inferiores a 2 segundos, e gostaríamos também de fazer ligeiras mudanças à estruturação das APIs de alguns dos módulos, para no futuro, ser mais fácil de responder a novas queries caso fosse necessário.

Apêndice A

Grafo de Dependências

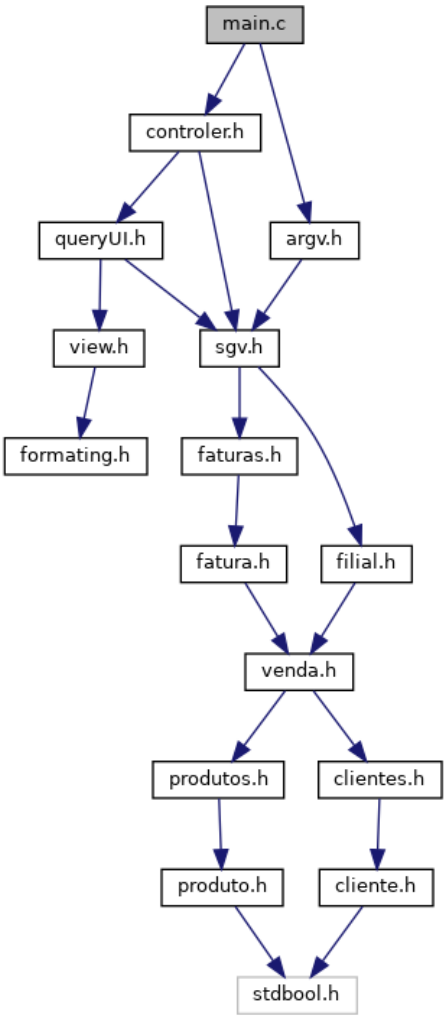

Figura A.1: Grafo de Dependências do projeto gerado pela ferramenta *Doxygen*

Apêndice B

Uso de Memória

Figura B.1: Gráfico de uso de memória

Apêndice C

Tabela de Tempos de Execução

	1 Milhão	3 Milhões	5 Milhões
Load Time	3.30	8.88	14.22
Query 2	0.003	0.003	0.003
Query 3	0.00001	0.00003	0.00001
Query 4	0.004	0.003	0.004
Query 5	0.003	0.003	0.003
Query 6	0.0004	0.0003	0.0003
Query 7	0.00007	0.0001	0.00008
Query 8	0.000009	0.000008	0.000007
Query 9	0.00002	0.00002	0.000007
Query 10	0.0003	0.00005	0.00005
Query 11	0.194	0.193	0.194
Query 12	0.00003	0.00003	0.00003

Tabela C.1: Tempo (em segundos) das queries para um dado número de vendas